Optimised MPI for HPEC applications

Gérard Cristau

Thales Computers
3100 Spring Forest Road
Raleigh, NC 27616
Tel: (919) 231-8000
Fax: (919) 231-8001

Gerard.Cristau@thalescomputers.fr
(France)

We propose an implementation of the MPI standard, suitable for HPEC applications.

The HPEC requirements addressed by the proposed implementation include:
- support of "zero copy" transport mechanisms
- memory management
- handling heterogeneous communication topologies

Zero copy communications are key for HPEC, in order to allocate memory bandwith to actual communications rather than to data transfers between buffers. The semantics and implementation issues associated with the mapping of MPI primitives on zero copy mechanisms such as VIA [1] or RDMA [2] have been discussed [3, 4, 5] and the feasibility and performance benefits have been demonstrated on the basis of popular MPI implementations such as LAM [6] or MPICH [7].

Optimised, industrial MPI implementations based on zero copy transport should be provided on the high-performance, low latency media suitable for HPEC systems. The implementation described in this paper uses a serial RapidIO network across PowerPC computing nodes. The applicability to Infiniband or Gigabit ethernet is also discussed.

Zero copy communications implies constraints on memory allocation and buffer management; memory buffer management is neither covered nor precluded by the MPI standard. Defining an API allowing the needed level of memory buffer management, while preserving compatibility with the MPI standard is one of the issues addressed by this implementation.

The paper finally proposes a mechanism, compatible with the MPI standard, to describe the communication topology of a HPEC software, such as a dataflow, signal processing application. Such an application is described as a set of tasks interconnected by virtual links. The application graph, and its mapping on a possibly heterogeneous computing and communication infrastructure, is defined through an independent configuration file, allowing to modify the application deployment without recompilation. The translation of the application graph in terms of MPI groups and communicators is carried out at run-time, and is transparent to the user.

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE **01 FEB 2005**	2. REPORT TYPE **N/A**	3. DATES COVERED **-**
4. TITLE AND SUBTITLE **Optimised MPI for HPEC applications**		5a. CONTRACT NUMBER
		5b. GRANT NUMBER
		5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)		5d. PROJECT NUMBER
		5e. TASK NUMBER
		5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Thales Research & Technology Thales Computers**		8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		10. SPONSOR/MONITOR'S ACRONYM(S)
		11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release, distribution unlimited

13. SUPPLEMENTARY NOTES
See also ADM00001742, HPEC-7 Volume 1, Proceedings of the Eighth Annual High Performance Embedded Computing (HPEC) Workshops, 28-30 September 2004 Volume 1., The original document contains color images.

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**UU**	**28**	

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std Z39-18

References

[1] http://www.viarch.org

[2] http://www.rdmaconsortium.org

[3] Jiuxing Liu, Jiesheng Wu, Sushmitha P. Kini, Pete Wyckoff
High performance RDMA-based MPI implementation over InfiniBand
Proceedings of the 17th annual international conference on Supercomputing
San Francisco, CA, USA, 2003

[4] R. Dimitrov and A. Skjellum.
An Efficient MPI Implementation for Virtual Interface (VI)
Architecture-Enabled Cluster Computing. http://www.mpi-softtech.com/publications/, 1998.

[5] Olivier Aumage, Guillaume Mercier.
MPICH/MADIII: a Cluster of Clusters Enabled MPI Implementation.
In 3rd {IEEE/ACM} International Symposium on Cluster Computing and the Grid
Tokyo, Japan, May 2003.

[6] Olivier Aumage, Guillaume Mercier, and Raymond Namyst.
MPICH/Madeleine: a true multi-protocol MPI for high-performance networks.
In Proc. 15th International Parallel and Distributed Processing Symposium (IPDPS 2001)
San Francisco, April 2001.

Abstract

Agenda

Optimised MPI for HPEC applications

- Middleware Libraries and Application Programming Interfaces
- Software Architectures, Reusability, Scalability, and Standards

Systems used for Dataflow applications

- Computing power requirements no evenly spread
- Various transport medium may coexist
- Need for QoS type behaviour
- Performance requirement for I/O between nodes

Requirements

- Need to map process to computing node
- Need to select specific link between process
- Need to implement zero-copy feature

- PROs
  - Available on almost every parallel/cluster machine
  - Ensures application code portability
- CONs
  - Made for collective parallel apps, not distributed apps.
  - No choice of communication interface (only know receiver)
  - Does not care about transport medium
  - No control on timeouts
  - Not a communication library
    (no dynamic connection, no select feature)

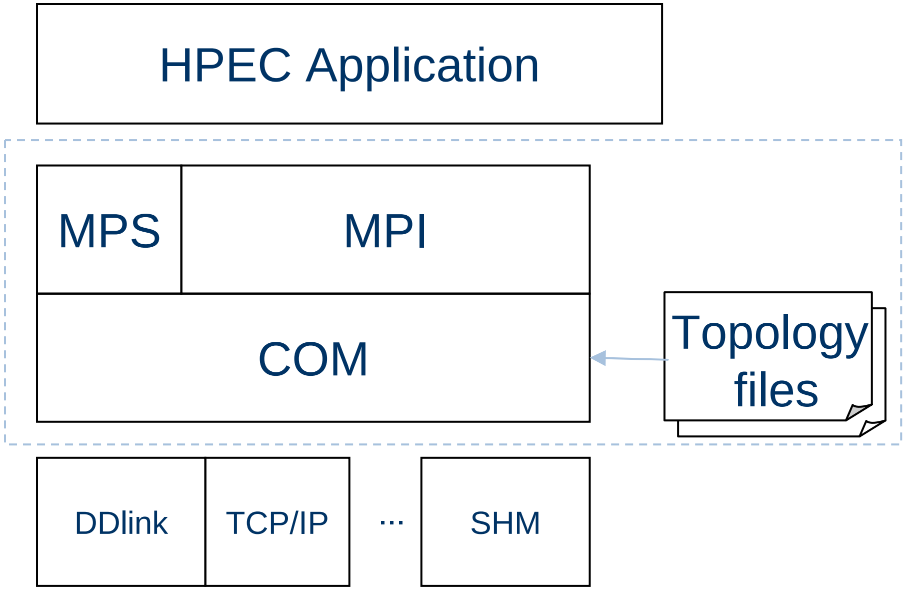
Zero-copy means memory management

- Same memory buffer used by application and I/O system
- At any given time, buffer must belong to application OR I/O

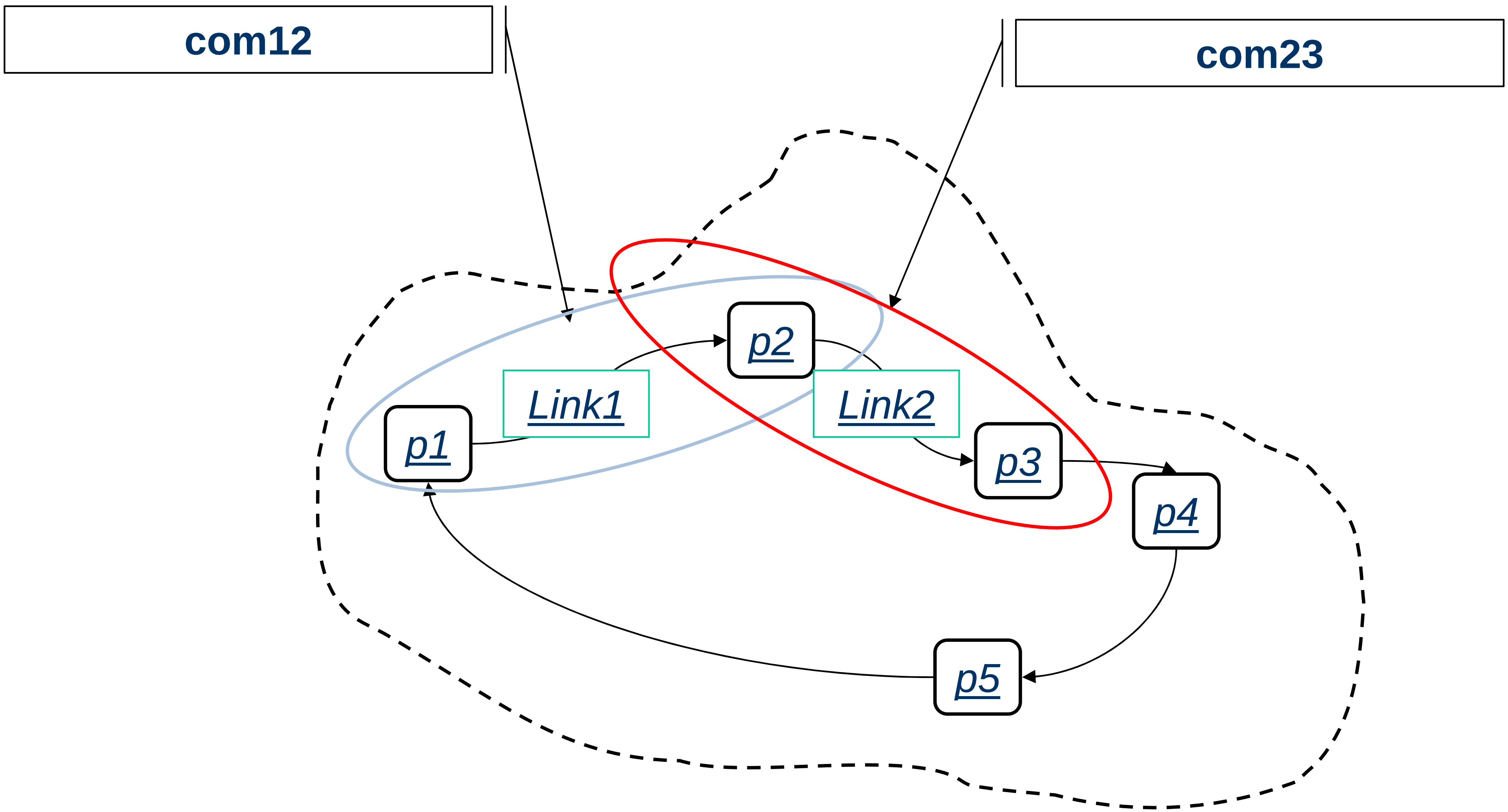
Zero-copy API

- Buffer Get
  - Data buffer now part of application data
  - Can be used as any private memory
- Buffer Release
  - Data buffer is not to be modified by application any more
  - Can be used by I/O system (likely hardware DMA)

- MPI Services (MPS) side to side with MPI
  - MPI application source portability
  - Links/Connector relationship
  - Real-Time support
    - Links to select communication channels (~ QoS)
    - Requests timeout support
  - Real zero-copy transfer
    - Buffer Management API (MPS)
  - Heterogeneous machine support
    - Topology files outside application

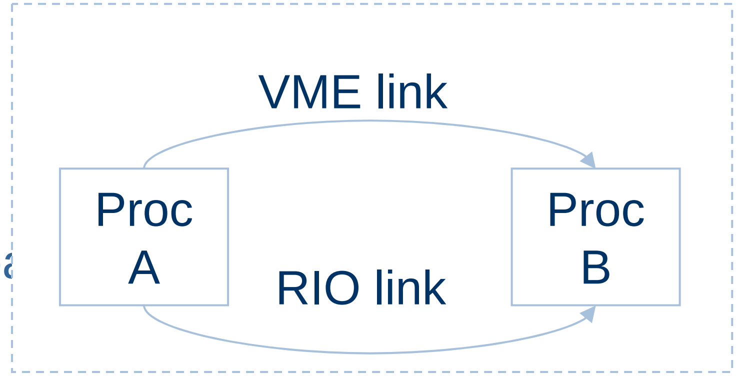


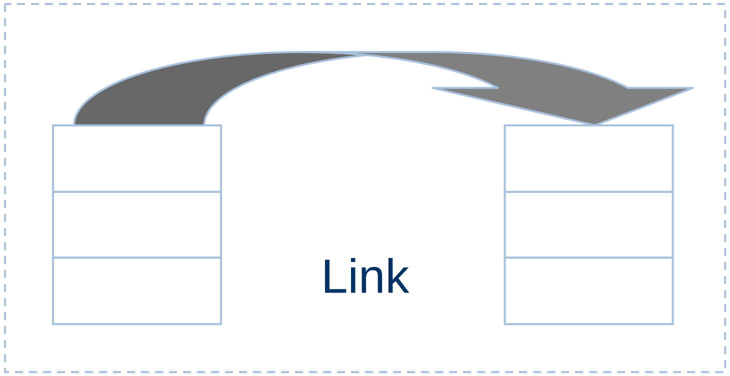
Dedicated MPI Communicator for Zero-copy Link



MPI_COMM_WORLD

- System topology described outside the application code
- External ASCII files with:
  - Process
    - Process name
    - Process Hardware location (boa
  - Link
    - Link name
    - Medium type (+medium-specific parameters)
    - Buffer size
    - Buffer count





MPS_Channel_create

(*chan_name, * rendpoint, MPI_Comm *comm, int *lrank, int *rrank) ;

```
link name     ^           ^             |              |          |
          remote end name |             |              |          |
specific communicator for the link      v              |          |
                   my rank in new communicator         v          |
                       remote end rank in new communicator        v
```

MPS_Process_get_name (int rank, char *name) ;

```
rank in MPI_COMM_WORLD     ^          |
          my name in link/process file v
```

MPS_Process_get_rank (char *name, int *rank) ;

```
name in link/process file      ^          |
   my  rank in MPI_COMM_WORLD             v
```

MPS_Buf_pool_init

```
    (MPI_Comm com,  way, * p_bufsize, * p_bufcount,  *p_mps_pool)
MPI communicator     ^      ^            |             |            |
     Send or Receive        |            v             v            |
                              buffer size & count                   v
                                                          MPS pool handle
```

MPS_Buf_get (p_mps_pool, void **p_buffer)

get buffer from pool (may block, or return EEMPTY)

MPS_Buf_release (p_mps_pool, void *buffer)

give buffer to I/O system (compulsory at each use) busy???

MPS_Buf_pool_finalize (p_mps_pool)

free all buffers, all coms must have completed first

MPI/MPS example

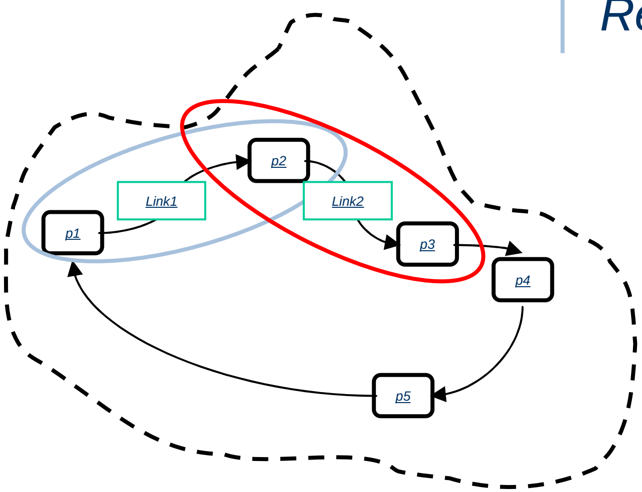
Create Dedicated Link
Get Specific connector
Initialize memory pool

```
MPI_Init(&argc, &argv);

MPS_Channel_create("link1", "proc2", &com, &lrank, &rrank);
MPS_buf_pool_init(com,  (sender) ? MPS_SND : MPS_RCV, &bufsize, &bufcount, &pool);
if (sender) {
        MPS_Buf_get(pool, &buf);
        Fill in with data
        MPI_Isend(buf, size/sizeof(int), MPI_INT, rrank, 99, com, &req);
        MPI_Wait(req, &status);
        MPS_Buf_release(pool, buf);
} else {
        …
}
MPS_Buf_pool_finalize(pool);
MPI_Finalize();
```

Take buffer ownership

Send on connector

Release buffer



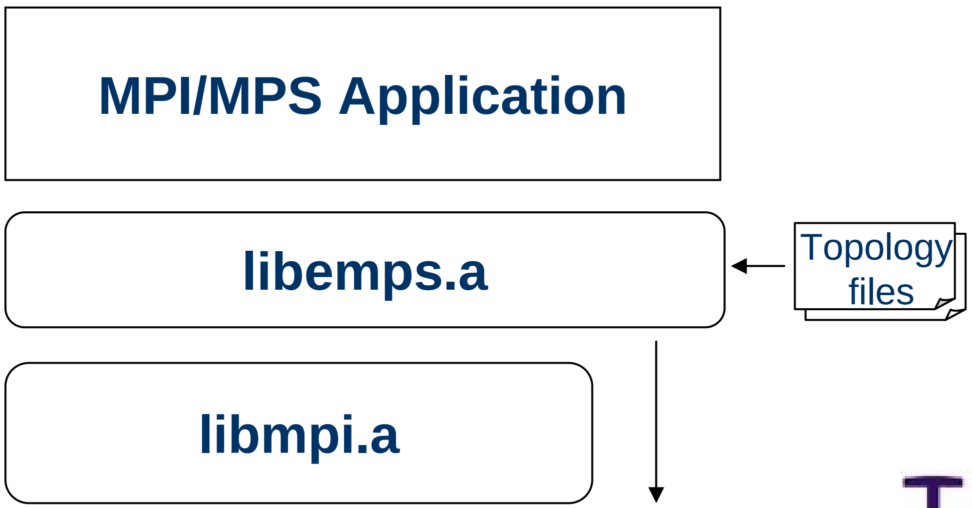
Portability

MPI application easily ported to MPI/MPS API

- See example

MPI/MPS application can run on any platform: EMPS

- EMPS is MPS emulation on top of standard MPI com
- Allow to run MPI/MPS code unmodified
  - Includes buffer and link management



Based on MICH ?? Version etc…

Software

- IA32 Red Hat, PowerPC LynxOS 4.0

HW Targets

- PC, Thales multiprocessor VME boards

Multi-protocol support in COM layer

- DDlink : Direct Deposit zero copy layer
  - Fibre Channel RDMA, Shared Memory, VME 2eSST, RapidIO
- Standard unix/posix I/O
  - Shared Memory, TCP/IP

Finalize process mapping

- MPI_RUN and HPEC compatible process mapping

Towards automatic code generation

- Create MPS / MPI code from HPEC application tools

More support for MPI-aware debug tools

- Like TotalView™

- Thank you
  vincent.chuffart@thalescomputers.fr

Poster

Agenda

Optimised MPI for HPEC applications

Benoit Guillon: Thales Research&Technology

Gerard Cristau: Thales Computers

Vincent Chuffart: Thales Computers

Systems used for Dataflow applications

- Computing power requirements not evenly spread
- Various transport medium may coexist
- Need for QoS type behaviour
- Performance requirement for I/O between nodes

Requirements

- Need to map process to computing node
- Need to select specific link between process
- Need to implement zero-copy feature

- PROs
  - Available on almost every parallel/cluster machine
  - Ensures application code portability
- CONs
  - Made for collective parallel apps, not distributed apps.
  - No choice of communication interface (only know receiver)
  - Does not care about transport medium
  - No control on timeouts
  - Not a communication library
    (no dynamic connection, no select feature)

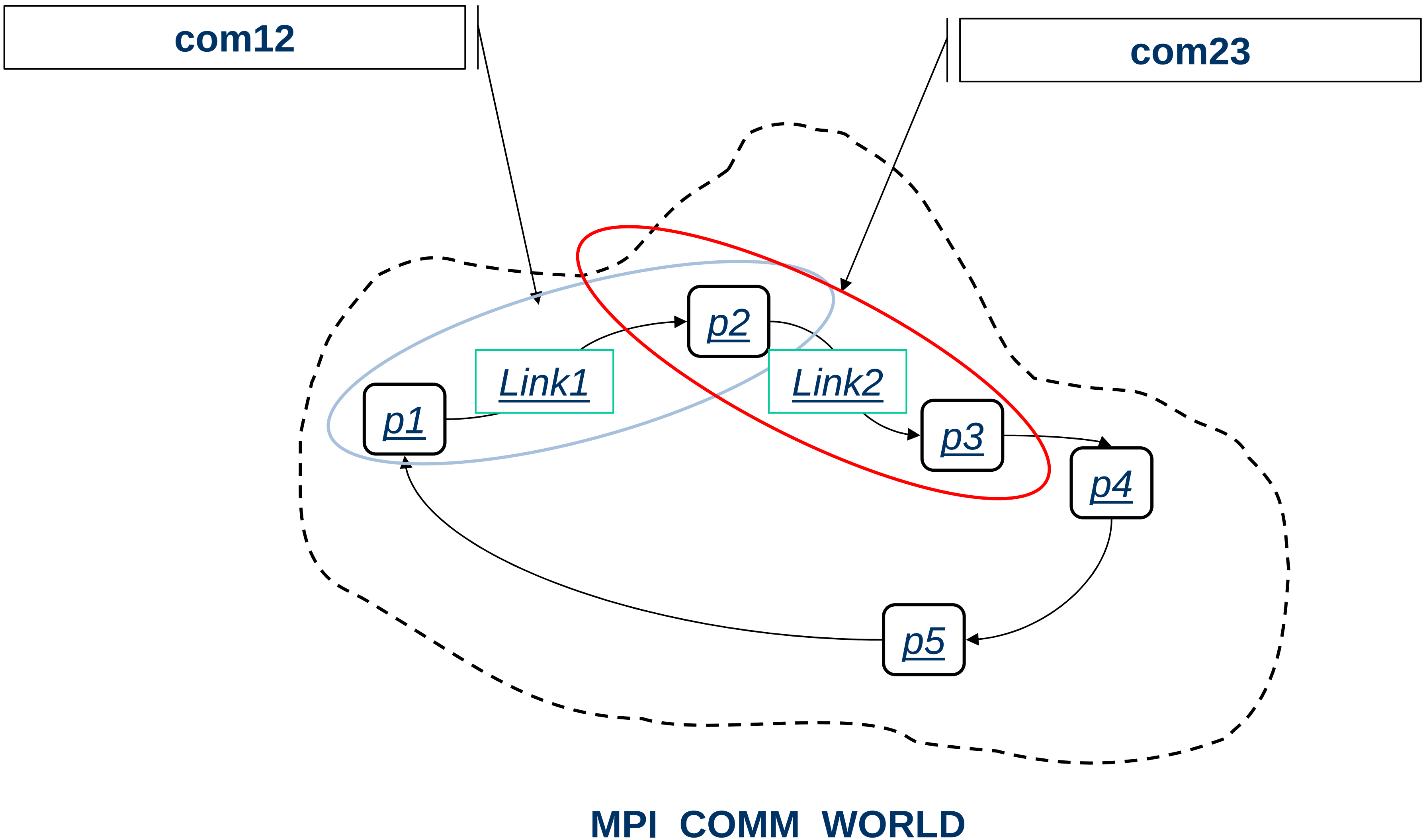
Zero-copy means memory management

- Same memory buffer used by application and I/O system
- At any given time, buffer must belong to application OR I/O

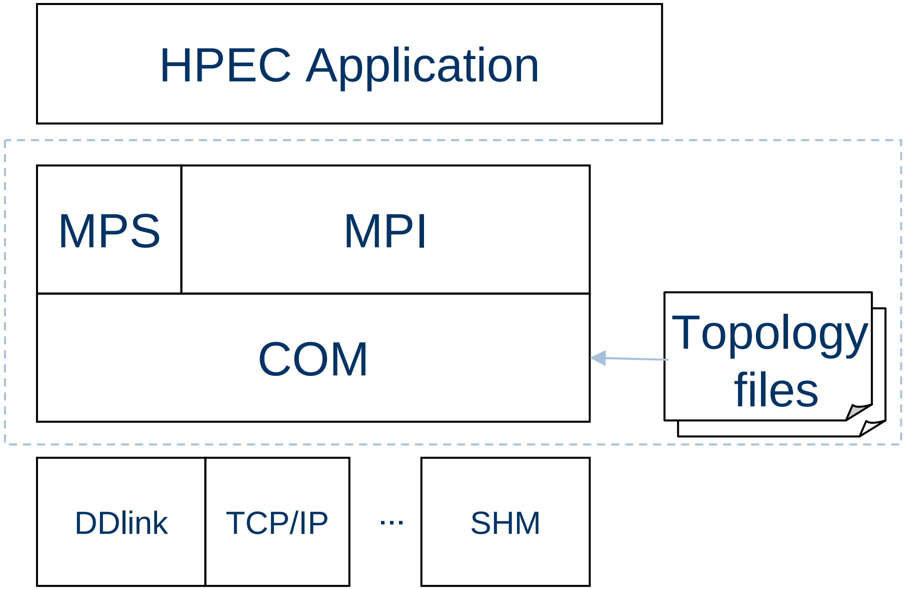
Zero-copy API

- Buffer Get
  - Data buffer now part of application data
  - Can be used as any private memory
- Buffer Release
  - Data buffer is not to be modified by application any more
  - Can be used by I/O system (likely hardware DMA)

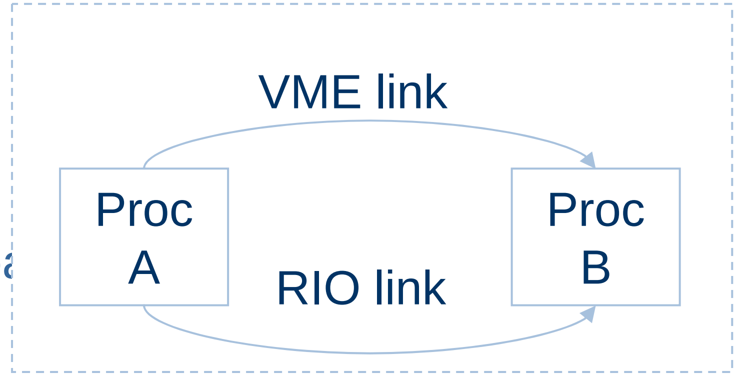
Dedicated MPI Communicator for Zero-copy Link

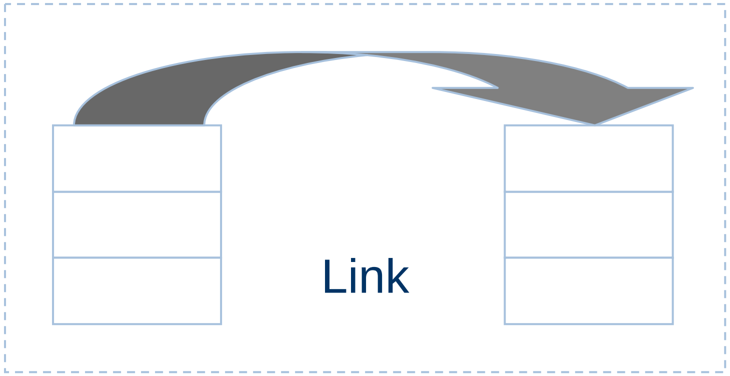


- MPI Services (MPS) side to side with MPI
  - MPI application source portability
  - Links/Connector relationship
  - Real-Time support
    - Links to select communication channels (~ QoS)
    - Requests timeout support
  - Real zero-copy transfer
    - Buffer Management API (MPS)
  - Heterogeneous machine support
    - Topology files outside application



- System topology described outside the application code
- External ASCII files with:
  - Process
    - Process name
    - Process Hardware location (boa
  - Link
    - Link name
    - Medium type (+medium-specific parameters)
    - Buffer size
    - Buffer count





MPS_Channel_create

(*chan_name, * rendpoint, MPI_Comm *comm, int *lrank, int *rrank) ;

link name ^ ^ | | |

remote end name | | | |

specific communicator for the link v | |

my rank in new communicator v |

remote end rank in new communicator v

MPS_Process_get_name (int rank, char *name) ;

rank in MPI_COMM_WORLD ^ |

my name in link/process file v

MPS_Process_get_rank (char *name, int *rank) ;

name in link/process file ^ |

my rank in MPI_COMM_WORLD v

MPS API: Buffers

MPS_Buf_pool_init

```
                (MPI_Comm com,  way, * p_bufsize, * p_bufcount,  *p_mps_pool)
 MPI communicator      ^      ^            |               |              |
       Send or Receive        |            v               v              |
                                      buffer size & count                 v
                                                                  MPS pool handle
```

MPS_Buf_get (p_mps_pool, void **p_buffer)

get buffer from pool (may block, or return EEMPTY)

MPS_Buf_release (p_mps_pool, void *buffer)

give buffer to I/O system (compulsory at each use) <u>busy???</u>

MPS_Buf_pool_finalize (p_mps_pool)

free all buffers, all coms must have completed first

MPI/MPS example

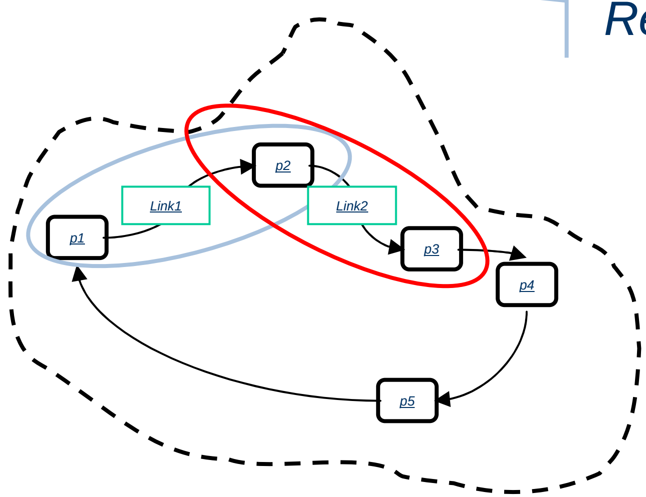
```
MPI_Init(&argc, &argv);

MPS_Channel_create("link1", "proc2", &com, &lrank, &rrank);
MPS_buf_pool_init(com,  (sender) ? MPS_SND : MPS_RCV, &bufsize, &bufcount, &pool);
if (sender) {
        MPS_Buf_get(pool, &buf);
        Fill in with data
        MPI_Isend(buf, size/sizeof(int), MPI_INT, rrank, 99, com, &req);
        MPI_Wait(req, &status);
        MPS_Buf_release(pool, buf);
} else {
        …
}
MPS_Buf_pool_finalize(pool);
MPI_Finalize();
```

Create Dedicated Link
Get Specific connector
Initialize memory pool

Take buffer ownership

Send on connector

Release buffer

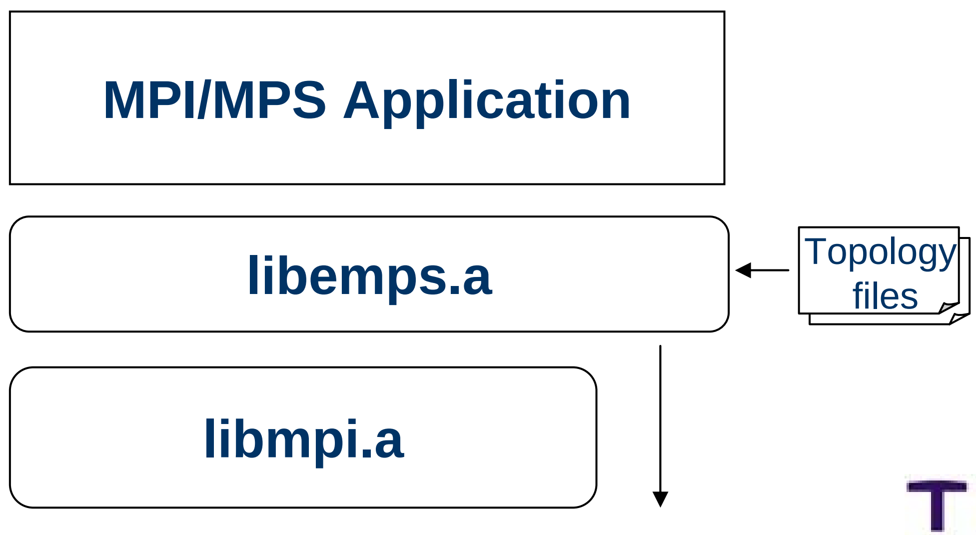


MPI application easily ported to MPI/MPS API

- See example

MPI/MPS application can run on any platform: EMPS

- EMPS is MPS emulation on top of standard MPI com
- Allow to run MPI/MPS code unmodified
  - Includes buffer and link management



Software

- Runs on IA32 Linux, PowerPC LynxOS 4.0

HW Targets

- PC, Thales Computers multiprocessor VME boards

Multi-protocol support in COM layer

- DDlink : Direct Deposit zero copy layer
  - Fibre Channel RDMA, Shared Memory, VME 2eSST, RapidIO
- Standard Unix/Posix I/O
  - Shared Memory, TCP/IP

Finalize process mapping

- MPI_RUN and HPEC compatible process mapping

Towards automatic code generation

- Create MPS / MPI code from HPEC application tools

- Thank you
  vincent.chuffart@thalescomputers.fr